Project 2.1: Random number generator.

Cruz Perez Gustavo Hazael

Superior School of Physics and Mathematics, Mexico City, Mexico.

Email: haxaleg@gmail.com

1. **Equations solved.**

Generation of pseudo-random numbers.

Formally a pseudo-random number generator is a structure where is a finite set of states is the $G = (X, x_0, T, U, g)$ *initial or seed state, the transformation is the transition function, is the finite set of possible observations, and is the output function*$Xx_0 \epsilon XT: X \to XUg: X \to U$.

For an $x_0$initial value, the sequence, provides a pseudo random number $x_n = T(x_{(n-1)})$ defined by a relation, $u_n$said sequence is finite because we define a as a $u_n = g(x_n)$ finite set, which will imply that for a certain number of steps you have for X some $x_j = x_i$ and from this index $j > i$ for everything $x_{j+k} = x_{(i+k)}$that is to say the generator has a certain period.$k \geq 0$

The period of a pseudo-random number generator.

Hull–Dobell theorem: A pseudorandom number generator has period equal to M if and only if the following conditions are met.

1) $M\ y\ c$ are cousins to each other
2) If q is a prime number that divides M, then q divides a $a - 1$
3) If 4 divides M, then 4 divides a $a - 1$

Checking uniformity.

The uniformity of a set of random numbers refers to deciding whether the generated numbers can be considered as a realization of a simple random sample of a distribution in the interval (0.1), that is, if the generated numbers are distributed equally in the interval.

Kolmogorov-Smirnov contrast.

Given a simple random set

$$\{x_1, x_2, \dots, x_n\}$$

Its empirical distribution function is.

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{x_i \leq x} = \frac{|valores\ x_i \leq x|}{n}$$

If F_0 it is the hypothetical or continuous distribution function, we have the following hypotheses.

$$Hipótesis\ nula \qquad H_0: F_n = F_0$$

$$Hipótesis\ alternativa \qquad H1: F_n \neq F_0$$

I mean.

$$H_0: El\ conjunto\ de\ números\ se\ distribuye\ uniformemente\ en\ el\ intervalo\ (0,1)$$

$$H_1: El\ conjunto\ de\ números\ \boldsymbol{no}\ se\ distribuye\ uniformemente\ en\ el\ intervalo\ (0,1)$$

The contrast statistic for the Kolmogorov-Smirnov goodness test is.

$$D_n = \sup_{x \epsilon \mathbb{R}} |F_n(x) - F_0(x)|$$

The distribution D_n is tabulated for certain significance values and a given number of data

2. Numerical method or algorithm used.

Method of linear congruences

Given a seed value of , $x_0$and integer values of , a, c, Meach x_n giver value by the following equation is considered a random number.

$$x_n \equiv ax_{(n-1)} + c\ mod\ M$$

To check uniformity using the Kolmogorov-Smirnov test, the following algorithm is followed. First we normalize the random numbers by dividing them by M, so that they are generated in the interval (0.1), we divide it among 10 classes and a count is made of how many points fall in each class.

Then the cumulative probability is calculated to be able to compare it with the ideal frequency that should present a uniform distribution, the absolute value of each subtraction of the accumulated probabilities for the 10 classes is calculated and we take the largest value, so we would have, that $D_n$we can compare using a table, where it tells us that for a data test where n with a significance of the $n \geq 50$ will be uniform yes.1%

$$D_n \leq \frac{1.63}{\sqrt{n}}$$

3. Code listing.

Table 1: Code listing

```
#Generador of random numbers
import matplotlib.pyplot as plt
import pylab as pl
from math import*
Import random

x0=int(input("Enter the value of x_0()seed: "))
a=int(input("Enter the value of a: "))
c=int(input("Enter the value of c: "))
M=int(input("Enter the value of M: "))
n=int(input("How many numbers do you want?"))
listrandom=[]


#inciso (a)
random def(x0,a,c,M):#generador of random numbers

for i in range(n):
x0=(a*x0+c)%M
listrandoms.append(x0)
listrandom[i]=listrandom[i]/M#divide by M ensures random numbers between 0
and 1
#print(listrandom[i])
random(x0,a,c,M)



#inciso (b)
def range(x0,a,c,M):
listal=[]
search=(a*x0+c)%M#Save the first number to find it again in the list
x0=(a*x0+c)%M
ran=0
var=False

while var==False:
x0=(a*x0+c)%M
listal.append(x0)
ran+=1
if search in listal:
var=True

print(f'\nThe range is = {ran}\n')
range(x0,a,c,M)




#inciso (c)
def graph1(listrandom):
listapar=[]
for i in range(len(listrandoms)):
if i%2==0:
```

```
listpar.append(listrandom[i])


odd list=[]
for i in range(len(listrandoms)):
if i%2!=0:
oddlist.append(listrandom[i])

pl.plot(evenlist,odd list,'b.')
pl.grid(True)
pl.title('Graph of pairs x_(2i-1) vs x_2i')
pl.xlabel('x_(2i-1)')
pl.ylabel('x_2i')
pl.show()
Graph1(ListRandom)



#inciso (d)
def graph2(listrandoms):
numbering=[]
for i in range(len(listrandoms)):
numbering.append(i+1)

pl.plot(numbering,listrandom,'b',linewidth=0.8)
pl.grid(True)
pl.title('Graph of x_i vs i')
pl.xlabel('i')
pl.ylabel('xi')
pl.show()
Graph2(listrandom)



#inciso (e)
print("Kolmogorov- Smirnov test For generator created")
def kolmogorov (listrandoms):
frequency=[]
for i in range(10):#Calcula the number of points in each of the 10 intervals
c=0
for j in range(len(listrandoms)):

if (I/10)<=listrandom[j]<=((I+1)/10):
c+=1
frequency.append(c)


freac=[]#Calcuar the cumulative frequency
a0=0
for i in range(len(frequency):
a=frequency[i]+a0
a0=a
freac.append(a)
for i in range(10):cumulative empirical #probabilidad
freac[i]=freac[i]/n
```

```
ideal=[]
for i in range(10):
ideal.append((i+1)/10)#probabilidad cumulative of a uniform distribution


subtraction=[]
for i in range(10):#restar empirica to uniform
subtract.append(abs(freac[i]-ideal[i]))


maximum=0
for i in range(10):
if subtract[i]>=maximum:
maximum=subtract[i]


D=1.63/sqrt(n)#para a significance value of 1%

if maximum<D:
print("Test result: H0\n is not rejected")
else:
print("Test result: H0\n is rejected")
kolmogorov(listrandom)



#inciso (d)
def comparisonpython(n):
print("Kolmogorov- Smirnov Test for Python Generator")
numbers=[]
for i in range(n):
x=random.random()
numbers.append(x)
kolmogorov(numbers)

ComparisonPython(n)
```

```

**4. Visualization.**

The code shown in Table 1 generates random numbers using the linear congruence method, and normalizes them to use the Kolmogorov-Smirnov goodness test. The following image shows the distribution of points for two different cases.
```

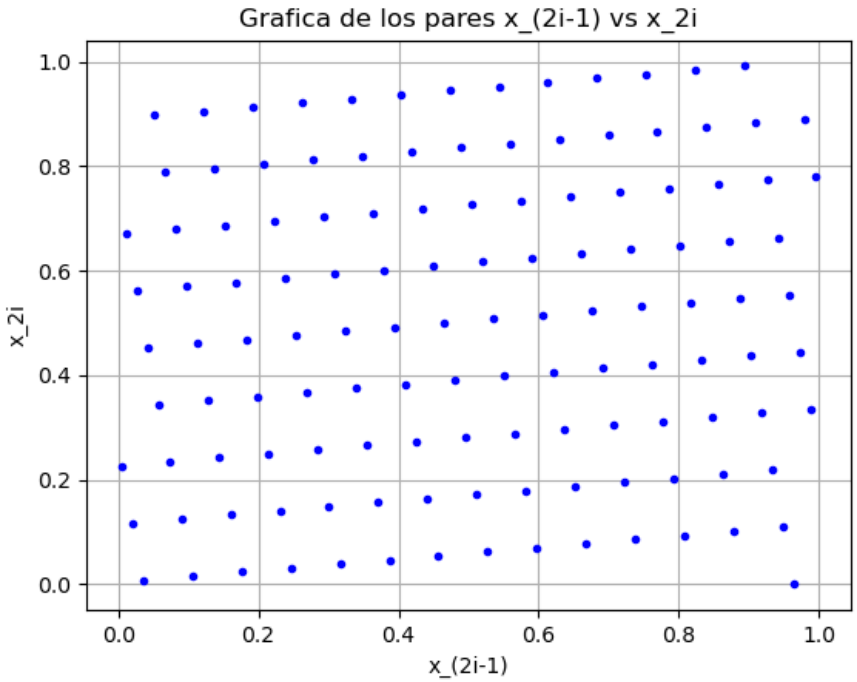

Image 1: Graphic of x_{2i-1} vs x_{2i} con $i = 1, \dots, M$

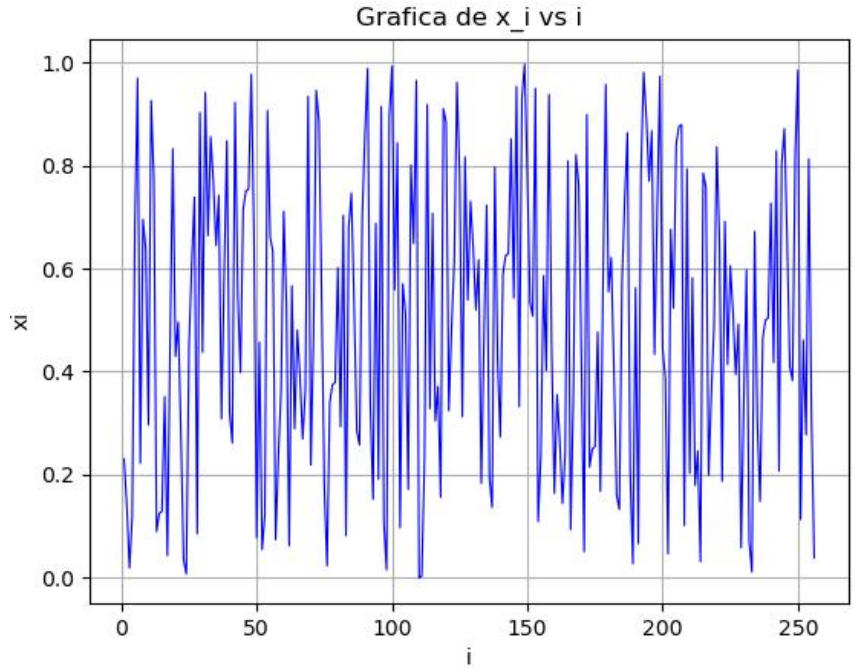

Image 2: Graph of the points x_i vs i

5. Discussion.

In image 1 we can see proof that the numbers are not entirely random because they follow an established order given by the input parameters.

We see that the Kolmogorov-Smirnov test can give us two possible results; that is not denied , that is to say that for a given data set it behaves as the hypothetical distribution, H_0 F_0in this case if the set of number is random it should be denied H_0. The other result of the test is that it is accepted that is to say that it is not uniform, and as we saw both the generated program and the python generator are uniform for a significance of the H_1 1%

6. **Classic.**

The values generated by the linear congruence method satisfy that.

$$x_n = a^n x_0 + c\frac{a^n - 1}{a - 1}\ mod\ m$$

As we can see the generated values are completelycharacterized by the input values so we can check something we already expected; the generator is not 100% random, which is something that all methods meet.

Another disadvantage is that our set of numbers can only take specific values$0, \frac{1}{M}, \frac{2}{M}, \dots, \frac{M-1}{M}$, and only for large values of M can we consider the set to behave as a sequence of a uniform continuous variable. These problems can be solved by refining generation methods, such as those generated by George Marsaglia.

7. **Bibliography.**

1) Manuel A. Pulido Cayuela. (2008). Item 1. Random number generation, University of Murcia, Website: https://webs.um.es/mpulido/miwiki/lib/exe/fetch.php?id=amio&cache=cache&media=wiki:simt1b.pdf
2) George C. Canavos. (1988). Probability and statistics: applications and methods. Mexico: McGraw Hill Latin America.
3) Rubin H. Landau, Manuel Paez, Cristian Constantin Bordeianu. (2012). Computational Physics Problem Solving with Computers. United States of : Wiley-VCH.